Volume: 2 Issue: XI Month of publication: November 2014

DOI:

www.ijraset.com Volume 2 Issue XI, November 2014
ISSN: 2321-9653
International Journal for Research in Applied Science & Engineering Technology (IJRASET)

Traversal of Trees

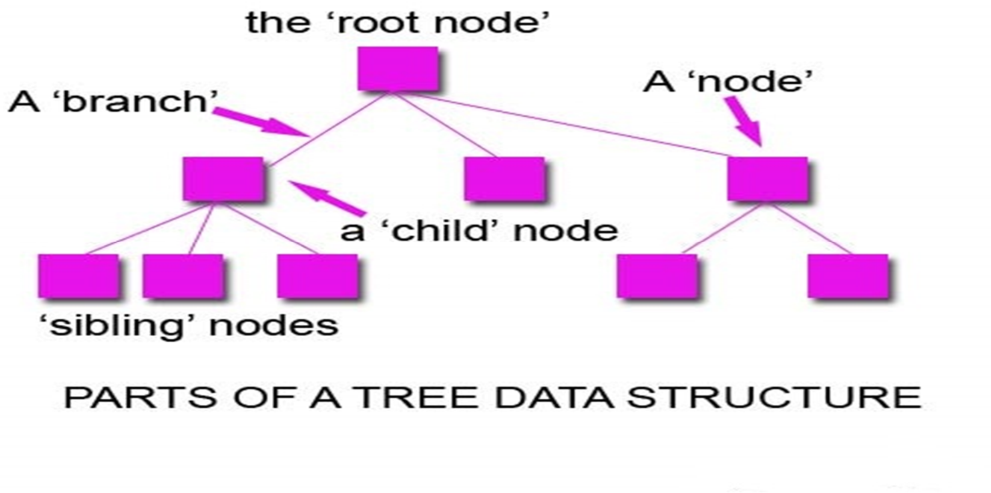
Gaurav Singh[1], Gitansh Arora[2], Lalit Kumar[3]
B.TECH, Dronacharya College of Engineering, Gurgaon, India

Abstract: To traverse a tree data structure is to process, however you like, every node in the data structure exactly once.The Traversal is that kind of process in which all nodes are visiting one bye one.in other words, this is the process which includes the process of visiting of all the nodes present in a tree. With the help of this process many process can be takes place in a tree. These processes are deletion of any element, insertion of any element, shifting of any element etc….. The traversal process plays an important role in these processes in very efficient manner. In a tree, there is a root node, left child, right child. The process of traversal includes the visiting of all these three nodes one by one.

Keywords: - Pre -order traveral, In-order traversal, post -order traversal.

I. INTRODUCTION

In all the data structures, one item follows another. Trees will be our first non-linear structure, lists, stacks, and queues are all linear structures. A tree is a collection of nodes. The collection can be empty; otherwise, a tree consists of a distinguished node r, called the root, and zero or more non-empty (sub) trees T_1, T_2, …, T_k each of whose roots are connected by a directed edge from r.In a tree, more than one item can follow another and the number of items that follow can vary from one item to another. Trees have many uses like representing family genealogiesas the underlying structure in decision-making algorithmsto represent priority queues (a special kind of tree called a heap)to provide fast access to information in a database (a special kind of tree called a b-tree). A binary tree is defined recursively: it consists of a root, a left subtree, and a right subtree.To traverse (or walk) the binary tree is to visit each node in the binary tree exactly once. Tree traversals are naturally recursive.



A. Types of tree traversal

A tree consists of three parts. A root node,a right node, and a left node. The types of the traversal is divide on the basis of the process of visiting the nodes of a tree. Tree traversal is a process of moving through a tree in a specified order toprocess each of the nodes. Each othe nodes is processed only once (although it may be visited more than once). Usually, thetraversal processis used to print out the tree. Traversal is like searching the tree except that in traversal the goal is tomove through the tree in some particular order. In addition, all nodes areprocessed in the traversal but searches cease when the required node is
Found.

If the order of traversal is not specified and the tree contains n nodes, then the number of paths that could be taken through the n nodes would be n factorial and therefore the information in the tree would be presented insome format determined by the path. Since there are many different paths, no real uniformity would exist in the presentation of information.

Therefore, three different orders are specified for tree traversals. These

Are called:

- Pre-order
- In-order
- Post-order

The traversals considered here are for a binary tree or a binary search tree. Because the definition of a binary tree is recursive and defined in terms ofthe left and right subtrees and the root node, the choices for traversalscan also be defined from this definition. In pre-order traversals, eachnode is processed before (pre) either of its sub-trees. In in-order, each
node is processed after all the nodes in its left sub-tree but before any ofthe nodes in its right subtree (they are done in order from left to right).In post-order, each node is processed after (post) all nodes in both of itssub-trees. Each order has different applications and yields different results.Consider the tree shown below (which has a special name - an expression
Tree):

```
        *
       / \
      /   \
     /     \
    +       +
   / \     / \
  /   \   /   \
 a     b c     7
```

The following would result from each traversal

- Pre-order: *+ab+c7
- In-order: a+b*c+7
- Post-order: ab+c7+*

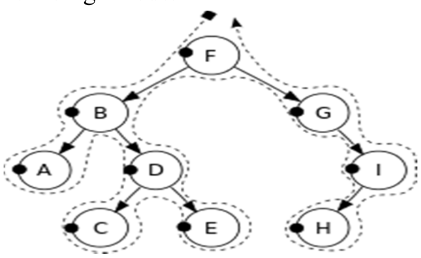
Notice that we can get all three common orders from the tree as compared tousing queues and a stack to get from in-fix to post-fix.

1) *Pre-Order Traversal:* In some ways this is the simplest to understand initially. However, eventhough each node is processed before the subtrees, it still requires thatsome information be retained while moving down the tree. In the exampleabove, the * is processed first, then the left sub-tree followed by the right subtree. Therefore, processing must return to the right sub-tree. After finishing the processing of the left subtree.

In the examples given, suppose the the processing involved in the traversal is to print out the information stored in each node. Since movement is down the left side of the tree beginning at the root, information on what is to the right must be kept in order to complete thetraversal of the right side of any subtree. The obvious ADT for such information is a stack. Because of its LIFO structure, it will be possibleto get the information about the right subtrees back in the reverse order in which it was encountered. In case of pre-order traversal, the first node always root node,

The process of doing a pre-order traversal iteratively then has thefollowing steps (assuming that a stack is available to hold pointers to the appropriate nodes):

a) Take the root node
b) Start at the root and begin execution
c) Write the root node
d) Take the left node of root and write the left node
e) Take the right node of root and write the right node.



Pre-order: F, B, A, D, C, E, G, I, H

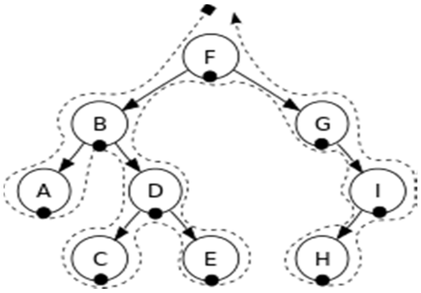
2) In-Order Traversal

The process is also interesting as post order traversal. The process of visiting of all nodes in this in-order traversal is quiet different from post order traversal. As the name indicates, the root node takes in between the left and right nodes in in-order traversal. In this Process, we will take left node form the root node firstly, after then, we will take the root node itself and I finally, we will take the right node from the root node.

This process when implemented iteratively also requires a stack and a boolean to prevent the execution from traversing any portion of a tree twice.The general process of in-order traversal follows the following steps:

a) Take the root node and print the left element of this node.
b) Write the root node itselt.
c) Write the right element from the root node element

The in-order traversal process can be easily understood by the given figure below:



In-order: A, B, C, D, E, F, G, H, I

3) Post-Order Traversal

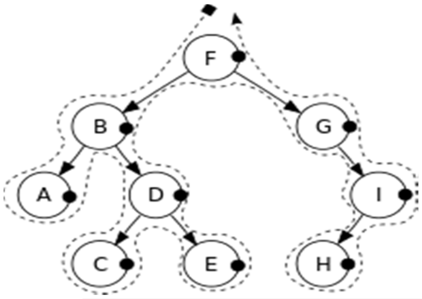
The Last and final tree traversal through we can visit Ann nodes of a tree, is Post-order traversal. This post-order traversal is totally different from the above two traversals that we have already discussed. The difference in takes place in the process of visiting the nodes of a tree.In this case, intead of root node, we will take the left node at first. As the name indicates, in case of post-order traversal, we will put the root node at the last position i.e. we will take root node at the end of the process in this type of traversal.

Iterative postorder traversal is discussed which is more complex than the other two traversals (due to its nature of non-tail recursion, there is an extra statement after the final recursive call to itself). The postorder traversal can easily be done using two stacks though. The idea is to push reverse postorder traversal to a stack. Once we have reverse postorder traversal in a stack, we can just pop all items one by one from the stack and print them, this order of printing will be in postorder because of LIFO property of stacks.

Now the question is, how to get reverse post order elements in a stack – the other stack is used for this purpose.. If take a closer look at this sequence, we can observe that this sequence is very similar to preorder traversal. The only difference is right child is visited before left child and therefore sequence is “root right left” instead of “root left right”.The steps that are involved in this process are

a) Take the root node of tree
b) Write the left node of this root node
c) Write the right node of the root node
d) Finally, right the root node itself.

We can easily understand this process by a suitable example which is given below:



Post-order: A, C, E, D, B, H, I, G, F

B. When to use pre-order, in-order & post-order strategy

The traversal strategy the programmer selects depends on the specific needs of the algorithm being designed. The goal is speed, so pick the strategy that brings you the nodes you require the fastest.

1) If you know you need to explore the roots before inspecting any leaves, you pick pre-order because you will encounter all the roots before all of the leaves.
2) If you know you need to explore all the leaves before any nodes, you select post-order because you don't waste any time inspecting roots in search for leaves.
3) If you know that the tree has an inherent sequence in the nodes, and you want to flatten the tree back into its original sequence, than an in-order traversal should be used. The tree would be flattened in the same way it was created. A pre-order or post-order traversal might not unwind the tree back into the sequence which was used to create it.

REFRENCES

[1] Discrete Mathematics by : *S.B. Gupta*
[2] Data Structures by: *A.K. Sharma*
[3] http://www.wikipedia.org.com